DOCUMENT RESUME

ED 341 558 SE 052 486

AUTHOR Greenwell, Raymond N.
TITLE Pascal: Solutions to Projects. Dissemination Packet--Summer 1989: Booklet #5.
INSTITUTION Hofstra Univ., Hempstead, NY. Dept. of Mathematics.; Hofstra Univ., Hempstead, NY. School of Secondary Education.
SPONS AGENCY National Science Foundation, Washington, D.C.
PUB DATE 89
CONTRACT TEI8550088,8741127
NOTE 55p.; For related documents, see SE 052 482-490.
PUB TYPE Guides - Classroom Use - Instructional Materials (For Learner) (051) -- Guides - Classroom Use - Teaching Guides (For Teacher) (052) -- Computer Programs (101)

EDRS PRICE MF01/PC03 Plus Postage.
DESCRIPTORS *Computer Assisted Instruction; *Computer Software Evaluation; Higher Education; High Schools; *Inservice Teacher Education; *Mathematics Education; *Mathematics Teachers; Secondary School Mathematics; Secondary School Teachers; Teacher Education Programs; Teacher Workshops
IDENTIFIERS *Hofstra University NY; *PASCAL Programing Language

ABSTRACT

This booklet is the fifth in a series of nine from the Teacher Training Institute at Hofstra University (New York) and contains the instructor's and several participants' course project solutions for both Pascal programing courses within the institute. Included in this booklet are Pascal programs utilized with the following project topics: (1) combinatoric strategies and Pascal's triangle; (2) synthetic division algorithms; (3) recursive generation of binomial coefficients; (4) studies of permutations; (5) complex numbers in the process of matrix multiplication; (6) producing tree diagrams with integers; and (7) sorting routines and linked lists. (JJK)

****
* Reproductions supplied by EDRS are the best that can be made *
* from the original document. *
****

SE

ED341558

HOFSTRA UNIVERSITY

TEACHER TRAINING INSTITUTE

Department of Mathematics and School of Secondary Education
Hofstra University
Hempstead, NY 11550

DISSEMINATION PACKET - SUMMER 1989

Booklet #5

RAYMOND N. GREENWELL

PASCAL: SOLUTIONS TO PROJECTS

---

*NSF Grant # TEI*8550088, 8741127

U.S. DEPARTMENT OF EDUCATION
Office of Educational Research and Improvement
EDUCATIONAL RESOURCES INFORMATION CENTER (ERIC)

X This document has been reproduced as received from the person or organization originating it.

☐ Minor changes have been made to improve reproduction quality.

• Points of view or opinions stated in this document do not necessarily represent official OERI position or policy.

"PERMISSION TO REPRODUCE THIS MATERIAL HAS BEEN GRANTED BY
Raymond Greenwell

TO THE EDUCATIONAL RESOURCES INFORMATION CENTER (ERIC)."

BEOS2486

BEST COPY AVAILABLE

This booklet is the fifth in a series of nine booklets which constitute the Hofstra University Teacher Training Institute (TTI) packet. The Institute was a National Science Foundation supported three-year program for exemplary secondary school mathematics teachers. Its purpose was to broaden and update the backgrounds its participants with courses and special events and to train and support them in preparing and delivering dissemination activities among their peers so that the Institute's effects would be multiplied.

This packet of booklets describes the goals, development, structure, content, successes and failures of the Institute. We expect it to be of interest and use to mathematics educators preparing their own teacher training programs and to teachers and students of mathematics exploring the many content areas described.

"Problem Solving via Pascal" was an introductory course in Pascal programming, with an emphasis on solving problems related to the high school mathematics curriculum. "Problem Solving via Pascal Data Structures" was a followup course covering those topics in the Advanced Placement Computer Science curriculum not discussed in the first course.

This booklet contains an extensive sampler of instructor and participant project solutions for both Pascal courses. Projects include: solutions of triangles using the Laws of Sine

and Cosine, Pascal's triangle, synthetic division of polynomials, permutations, matrix multiplication, trees and linked lists.

TEACHER-TRAINING INSTITUTE REPORT #5:

PASCAL: SOLUTIONS TO PROJECTS

by

Raymond N. Greenwell
Mathematics Department
Hofstra University
Hempstead, NY 11550

copyright (c) Raymond N. Greenwell
All rights reserved, except that permission will be granted to make a limited number of copies for noncommerical educational purposes, upon written request, provided that this copyright notice shall appear on all such copies.

TABLE OF CONTENTS

A. Combinatorics and Pascal's Triangle-Robert Silverstone

B. Sythetic Divison 1-Joyce Bernstein

C. Sythetic Division 2

D. Binomial Coefficients Recursively

E. Permutations 1-Carolyn Walters

F. Permutations 2

G. Permutations Afternote

H. Multiplying Matrices 1 (project description and solution)-Carolyn Walters

I. Multiplying Matrices 2-Frances Marcello

J. Trees (project description and solution)-Carolyn Walters

K. Linked Lists (project description and solution)-Carolyn Walters

Robert Silverstone
Hofstra University
Math 287
Prof. Greenwell

Program_ COMBINATORICS & PASCAL'S TRIANGLE _Number_ 6 __ Date_ 12/6 __

ABSTRACT:

The number of combinations of n objects taken r at a time is symbolized by nCr and is computed by the formula:

nCr = n! / (r! * (n-r)!)

where k! = 1 * 2 * ... * (k-1) * k = k * (k-1)!

Three methods are used in this program to calculate nCr.

The first is by iteration: It calculates the accumulated product (n/r) * (n-1/r-1) * ... * (n-r+1/1)

The second method finds, using recursion, the value of k! (k = n,r,n-r) by the relation k! = k * (k-1)! ,and then calculates nCr by nCr = n! / (r! * (n-r)!)

The third method uses the idea that nCr is the r+1 st element of the n+1 st row of the PASCAL triangle, and the triangle has as its defining property:

nCr = n-1Cr-1 + n-1Cr

and the computation is again accomplished by using recursion.

Recursion is the process by which a function or a procedure calls upon itself.

The second program computes all of the rows of PASCAL TRIANGLE up to row N, which is entered from the keyboard.
Row K of the pascal triangle consists of the numbers

kCr, for r = 0,...,k

The process that is used to calculate kCr is METHOD1.

PROCEDURES and FUNCTIONS:

HEADING.....Gives a description of the program

METHOD1.....Calculates nCr by the first method above.

METHOD2.....Calculates nCr by the second method above.

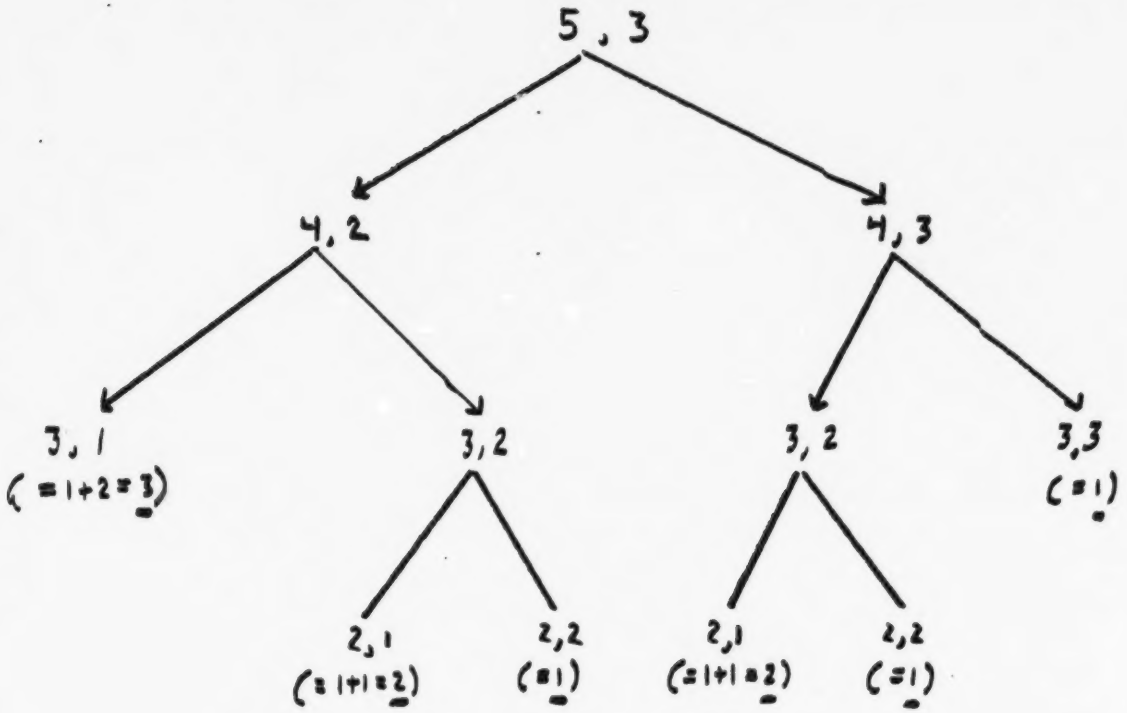
METHOD3.....Calculates nCr by the third method above.

OPERATING PROCEDURES:

Run either NCR for the calculation of nCr or PASCAL for the triangle.

For nCr, type in first value, the SPACE, the second value and then RETURN.

Example of Method 2 for $_5C_3$



$\therefore \quad {}_5C_3 = 3 + 2 + 1 + 2 + 1 + 1 = 10$

```
VAR
  N,R,ANS :INTEGER;
  C       :CHAR;

BEGIN
  PAGE(OUTPUT);
  WRITELN ; WRITELN ; WRITELN ;
  WRITELN('Computation of nCr by three methods':58);
  WRITELN;
  WRITELN('Method 1....nCr = (n/r)(n-1/r-1)...(n-r+1/1)');
  WRITELN;
  WRITELN('Method 2....nCr = n!/(r!*(n-r)!)');
  WRITELN('                  where k! is computed by recursion');
  WRITELN;
  WRITELN('Method 3....nCr = n-1Cr-1 + n-1Cr');
  WRITELN('                  and is computed by recursion');
  WRITELN
END;                  { of procedure HEADING                    }

VAR
 I:INTEGER;
 A:REAL;
BEGIN
  A := 1.0;
  FOR I := 0 TO B-1 DO
      A := A*(T-I)/(B-I);;
  METHOD1 := TRUNC(A)
END;                  { of function  METHOD1                    }

BEGIN
  IF (K<=1) THEN METHOD2 := 1
  ELSE METHOD2 := K*METHOD2(K-1);
END;                  { of function METHOD2                     }

BEGIN
  IF (J=0) THEN METHOD3 := 1
     ELSE
       IF (J>1) AND (J<K) THEN METHOD3 := METHOD3(K-1,J-1) + METHOD3(K-1,J)
          ELSE
            IF (J=1) THEN METHOD3 := K
               ELSE IF (J<1) OR (J>K) THEN METHOD3 := 0
                 ELSE IF (J=K) THEN METHOD3 := 1
END;                  { of function METHOD3                     }
```

```
  HEADING;
  REPEAT
    WRITE('Enter n and r -> ') ; READLN(N,R);
    WRITELN;
    {***
        METHOD 1
                ***}
    WRITELN('Answer method 1 is -> ':49,METHOD1(N,R));
    {***
        METHOD 2
                ***}
    ANS := TRUNC( METHOD2(N) / ( METHOD2(R) * METHOD2(N-R) ) );
    WRITELN;
    WRITELN('Answer method 2 is -> ':49,ANS);
    WRITELN;
    {***
        METHOD 3
                ***}
    WRITELN('Answer method 3 is -> ':49,METHOD3(N,R));

    WRITELN ; WRITELN ;
    WRITE('Go again? (y or n)  ') ; READ(C);
  UNTIL (C = 'N') OR (C = 'n');
  WRITELN ; WRITELN ;
  WRITELN('BYE - BYE':44)
END.
```

Computation of nCr by three methods

Method 1....nCr = (n/r)(n-1/r-1)...(n-r+1/1)

Method 2....nCr = n!/(r!*(n-r)!)
where k! is computed by recursion

Method 3....nCr = n-1Cr-1 + n-1Cr
and is computed by recursion

```
Enter n and r -> 3 , 0
                          Answer method 1 is -> 1
                          Answer method 2 is -> 1
                          Answer method 3 is -> 1

Go again? (y or n)  Y

Enter n and r -> 3 , 1
                          Answer method 1 is -> 3
                          Answer method 2 is -> 3
                          Answer method 3 is -> 3

Go again? (y or n)  Y

Enter n and r -> 3 , 2
                          Answer method 1 is -> 3
                          Answer method 2 is -> 3
                          Answer method 3 is -> 3

Go again? (y or n)  Y

Enter n and r -> 3 , 3
                          Answer method 1 is -> 1
                          Answer method 2 is -> 1
                          Answer method 3 is -> 1

Go again? (y or n)  Y

Enter n and r -> 0 , 0
                          Answer method 1 is -> 1
                          Answer method 2 is -> 1
                          Answer method 3 is -> 1

Go again? (y or n)  N
```

BYE - BYE

```
VAR
  N,I,J    : INTEGER;
  C        : CHAR;

  BEGIN
    WRITELN('COMPUTATION OF PASCAL TRIANGLE TO ROW N':59);
    WRITELN;
    WRITELN('This program will print out the PASCAL TRIANGLE');
    WRITELN('to row N, which is entered from the keyboard.');
    WRITELN('The method used is METHOD1 of the previous set.');
    WRITELN;
  END;                    { of procedure HEADING                    }

  VAR
   I:INTEGER;
   A:REAL;
  BEGIN
    A := 1.0;
    FOR I := 0 TO B-1 DO
        A := A*(T-I)/(B-I);
    METHOD1 := TRUNC(A)
  END;                    { of function  METHOD1                    }


  HEADING;
  REPEAT
    WRITE('Enter value for n -> '); READLN(N);
    FOR I := 0 TO N DO
      BEGIN
        WRITE(' ':30-5*I);
        FOR J := 0 TO N DO
            IF (METHOD1(I,J)<> 0) THEN WRITE(METHOD1(I,J):10);
        WRITELN;
      END;
    WRITE('Go again? (y or n)  ') ; READ(C);
  UNTIL (C = 'N') OR (C = 'n');
  WRITELN ; WRITELN ; WRITELN ;
  WRITELN('BYE - BYE':44);
END.
```

COMPUTATION OF PASCAL TRIANGLE TO ROW N

This program will print out the PASCAL TRIANGLE
to row N, which is entered from the keyboard.
The method used is METHOD1 of the previous set.

Enter value for n -> 5

```
                    1
                  1   1
                1   2   1
              1   3   3   1
            1   4   6   4   1
          1   5  10  10   5   1
```

Go again? (y or n) Y

Enter value for n -> 3

```
                  1
                1   1
              1   2   1
            1   3   3   1
```

Go again? (y or n) Y

Enter value for n -> 1

```
                  1
                1   1
```

Go again? (y or n) Y

Enter value for n -> 0

```
                  1
```

Go again? (y or n) N

BYE - BYE

```
Program syndiv(input,output);
{Joyce Bernstein}
{Inputs a polynomial, degree <= 10 and evaluates it at a given integer}
{using synthetic division}
Const
   Maxsize = 10;
Type
   Poly= array[0..maxsize] of integer;
Var
   dividend, addend,quotient:poly;
   eval,n:integer;
{**************procedure readpoly*****************************************}
Procedure Readpoly(var P:Poly;var deg:integer);
   Var i: integer;
      Begin
         write('Input degree of polynomial:  ');
         readln(deg);

            for i := deg downto 0 do
         begin
            writeln ('enter the coefficient of x^',i:1);
            readln (p[i]);
         end
      end;{readpoly}
{**************procedure readvalue****************************************}
procedure readvalue(var num:integer);
     Begin
        Write('For what integer value would you like the polynomial evaluated? '
);
        Readln(num);
     End{readvalue};
{**************procedure divide*******************************************
**}
procedure divide(c:poly;var a,b:poly;num,deg:integer);
   var i:integer;
   Begin
      A[deg] :=0;
      For  i:= deg downto 1 do
         begin
            b[i]:= c[i] + a[i];
            a[i-1]:=b[i]*num;
         end;
         b[0]:=c[0]+a[0]
    end;
{**************Procedure printresult**************************************}
procedure printresult(c,b:poly;num,deg:integer);
var i:integer;
   Begin
   writeln('This problem was solved using synthetic division.');
   writeln;
   begin
      For I:=deg downto 0 do
      write ('+',c[i],'X^',i);
   end;
      write (' divided by (x - ',num,') is ');
```

```
    begin
       for i := deg downto 1 do
       write('+',b[i],'X^',i-1);
    end;
    writeln;
write( 'The remainder-which is the value of the polynomial at ',num,' is ',b[0])
    end;
{***************main program*************************************************
***}
begin
   readpoly(dividend,n);
   if n > maxsize then
       Writeln('You must have degree <= ',maxsize)
       else
          Begin
             readvalue(eval);
             Divide(dividend, addend, quotient, eval, n);
             printresult(dividend,quotient,eval,n)
          end;
end.
```

```
program sytheticdiv (input, output);
{Using sythetic division, compute resulting polynomial and remainder}
{Written by Ray Greenwell for Math 287, Fall 1986}
 const
  maxdegree = 20;
 var
   i,    {a counter}
   n,    {the degree of the polynomial}
   r  {the point where we are evaluating the polynomial}
    : integer;
   a,    {the original polynomial}
   b {the quotient polynomial}
    : array[0..maxdegree] of integer;
   answer : char;
begin
 repeat
   write('Enter degree of polynomial: ');
   readln(n);
   while (n < 0) or (n > maxdegree) do
    begin
      writeln('Degree must be between 0 and 20.');
      write('Enter degree of polynomial: ');
      readln(n);
    end;{while}
   for i := n downto 0 do
    begin
      write('Enter coefficient of x^', i : 1, ': ');
      readln(a[i])
    end;{for i}
   write('Divisor is (x - r).  Enter value of r: ');
   readln(r);
{The next 3 lines are the synthetic division computation}
   b[n] := a[n];
   for i := n - 1 downto 0 do
    b[i] := b[i + 1] * r + a[i];
   writeln('Result =');
   for i := n downto 2 do
     write(b[i] : 1, '*x^', i - 1 : 1, ' + ');
   writeln(b[1] : 1);
   writeln('Remainder = ', b[0] : 1);
   write('Try again? (y/n) ');
   readln(answer);
 until answer <> 'y'
end.
```

```
program recursivebinom (input, output);
{Compute binomial coefficients two recursive ways.}
{Written for Math 287, Fall 1986, by Ray Greenwell}
 var
  n, r : integer; {The variables in "n choose r"}
  time1, time2, coefficient : longint;
  datetime : datetimerec;
{datetimerec is a nonstandard feature of Macintosh Pascal}

 function nCr1 (n, r : integer) : integer;
{Compute nCr knowing the value of (n-1)C(r-1)}
 begin
  if (r < 0) or (r > n) then
   nCr1 := 0
  else if r = 0 then
   nCr1 := 1
  else
   nCr1 := n * nCr1(n - 1, r - 1) div r
 end; {function nCr1}

 function nCr2 (n, r : integer) : integer;
{Compute nCr using Pascal's triangle}
 begin
  if (r < 0) or (r > n) then
   nCr2 := 0
  else if r = 0 then
   nCr2 := 1
  else
   nCr2 := nCr2(n - 1, r - 1) + nCr2(n - 1, r)
 end; {function nCr2}

 function timeinseconds : longint;
 begin
  timeinseconds := (datetime.hour * 60 + datetime.minute) * 60 + datetime.second
 end;

begin
 write('To compute the binomial coefficient nCr, please enter n and r: ');
 readln(n, r);
 write('Using the first method, the coefficient is ');
 gettime(datetime);
 time1 := timeinseconds;
 coefficient := nCr1(n, r);
```

```
  gettime(datetime);
  time2 := timeinseconds;
  writeln(coefficient : 1);
  writeln('Elapsed time= ', (time2 - time1) : 1, ' seconds.');
  write('Using the second method, the coefficient is ');
  gettime(datetime);
  time1 := timeinseconds;
  coefficient := nCr2(n, r);
  gettime(datetime);
  time2 := timeinseconds;
  writeln(coefficient : 1);
  writeln('Elapsed time= ', (time2 - time1) : 1, ' seconds.');
end.
```

```
Program Project5 (input,output);
{-------------------------------------------------------------}
{                                                             }
{             Carolyn Walters                                 }
{             Math 287                                        }
{             October 7, 1987                                 }
{             Project 5 - Permutations                        }
{                                                             }
{-------------------------------------------------------------}
{ This program asks the user to input a number from 1 to 26.  }
{ It will then print out the permutations of that number of   }
{ letters of the alphabet.  For example, if user puts in 2    }
{ the output would be   AB  BA, the permutations of the first }
{ two letters of the alphabet.  User is given the opportunity }
{ to run the program again or quit.                           }
{                                                             }
{-------------------------------------------------------------}

const    max = 26;          { number of letters in alphabet }
type     arytype = array[1..max] of char;

Procedure Directions;
begin
  writeln('This program will ask you to put in a number from 1 to 26.'
  writeln('It will then form all the permutations of the first');
  writeln('n ( n is the number you put in ) letters of the alphabet.')
  writeln;
end;
{------------------------ end directions -----------------------}

Procedure GetNum(var n:integer);
{ Allows user to put in the number of letters for the permutations. }

begin  { getnum }
   n := 0;
   while (n < 1) or (n > 26) do
   begin
      writeln('Please enter the number of letters from the alphabet');
      write('for this run of the program.');
      readln(n);
      writeln;
   end;   { while }
end;   { getnum }
{-------------------- end getnum ------------------------------}


Procedure PrintIt (a : arytype; var permnum : integer; n : integer);
{ Prints out the permutatuions and also counts them.  }

var  i : integer;
begin
    permnum := permnum + 1;
    if permnum < 100
       then write(' Permutation  ',permnum:2,'     ')
       else if permnum < 1000
            then write(' Permutation  ',permnum:3,'    ')
            else if permnum < 10000
                 then write(' Permutation ',permnum:4,'   ')
                 else write(' Permutation ',permnum:5,'  ');
    for i := 1 to n do
        write(a[i],' ');
    writeln;
end;
{-------------------- end printit ------------------------------}
```

```
Procedure Perm (n: integer);
{ The following procedure is not original.  It is adapted from }
{ a procedure taken from the following: Pascal Via Recursion,  }
{ by J.S. Rohl, Cambridge University Press, 1984, pp. 148-150. }
{ It takes an integer n and assigns the first n letters of the }
{ alphabet to an array.  It then forms the permutations of the }
{ letters.  It calls a procedure to print and count each.      }

var i,permnum : integer;
    a: arytype;                 { array of letters }

   Procedure Choose(d:integer);
   var i : integer;
       ch : char;               { needed for switching in array }
   begin
       ch := a[d];
       for i := d to n do
       begin
         a[d] := a[i];
         a[i] := ch;
         if d <> n then Choose(d+1) else PrintIt(a,permnum,n);
         a[i] := a[d];
       end;
       a[d] := ch;
   end;  { choose }
   {-------------------- end choose ----------------------}

begin  { perm }
   for i := 1 to n do
   a[i] := chr(ord('A') + i - 1);  { filling array }
   permnum := 0;
   Choose(1);
end;   { perm }
{------------------- end perm ---------------------------------}
```

```
Procedure Driver;
( Controls flow of program.  Gives user chance to run it again.)
var n : integer;
    response : char;
begin
   Directions;
   writeln;
   response := 'Y';
   while (response = 'Y') or (response = 'y') do
   begin
      GetNum(n);
      writeln;
      writeln('Here are the permutations of ',n:1,' elements:');
      writeln;
      Perm(n);
      writeln;
      repeat
         writeln;
         write('Do you wish to run this program again? Y or N: ');
         readln(response);
         writeln;
      until response in ['Y','y','N','n'];
      if response = 'N' or response =  'n'
         then begin
                writeln;
                writeln('Your decision to end.');
                writeln('Have a NICE day!!!');
              end;
   end;    ( while )
end;       ( driver )
{-------------------- end driver ------------------------------}

begin
  driver;
end.
```

```
program permutation (input, output);
{Write all permutations of the first n letters}
{Math 287 Project 5, Fall 1987}
{Written by Raymond Greenwell}
{Given the slick algorithm described on the other handout, this program}
{seems pretty clunky in comparison.  Nevertheless, it continues to be }
{requested by popular demand, so here it is.}
 type
  alphabet = 'A'..'Z';
  letterset = array[1..26] of alphabet;
 var
  letters : letterset;
  n, count : integer;

 procedure fill (var letters : letterset;
           n : integer);
{Fill the set letterset with the first n letters}
  var
   i : integer;
 begin
  for i := 1 to n do
   letters[i] := chr(ord('A') + i - 1)
 end; {procedure fill}

 procedure addto (var firstset, secondset : letterset;
           pos1, pos2, size1, size2 : integer);
{Add element pos2 of secondset, with size size2, to position pos1}
{of first set, with size size1, and remove the element from pos2}
{of secondset}
  var
   i : integer;
 begin
{first make room for the new element in set 1}
  for i := size1 downto pos1 do
   firstset[i + 1] := firstset[i];
  firstset[pos1] := secondset[pos2];
{close the gap in set 2}
  for i := pos2 to size2 - 1 do
   secondset[i] := secondset[i + 1];
 end;{procedure addto}

 procedure permute (newset, oldset : letterset;
           newn, oldn : integer);
```

```
(Recursive procedure which calls itself to print all permutations of)
(the newn letters in newset following the oldn letters in oldset)
  var
    i : integer;
 begin
  if newn = 1 then
   begin
    for i := 1 to oldn do
     write(oldset[i] : 1);
    writeln(newset[1] : 1);
    count := count + 1;
   end (then)
  else
   for i := 1 to newn do
    begin
     addto(oldset, newset, oldn + 1, i, oldn, newn);
     permute(newset, oldset, newn - 1, oldn + 1);
     addto(newset, oldset, i, oldn + 1, newn - 1, oldn + 1);
    end;(else)
 end;(procedure permute)

begin
 writeln('This program will write all permutations of the first n');
 writeln('letters of the alphabet.');
 write('Enter n: ');
 readln(n);
 count := 0;
 fill(letters, n);
 permute(letters, letters, n, 0);
 writeln('There are a total of ', count : 1, ' permutations.');
end.
```

The Case of the Mysterious Permutation Program

Many people have turned in very similar versions of the program to find all permutations of the first N letters of the alphabet, where N is entered by the user. The recursive subroutine goes something like this.

1) The subroutine is entered with a parameter L passed in. The idea is that the elements L through N are to be rearranged.
2) For K = L to N do steps 3 through 4 (or 5, if such a step is present).
3) Switch items K and L.
4) Call the subroutine with L increased by 1.
5) (The controversial step) Switch items L and K.

It was claimed by some that step 5 must be present or the program doesn't work. Unfortunately, the programs that included step 5 did not produce the permutations in alphabetical order. Others turned in programs that did not include step 5 but worked perfectly. How could this be? And why does this algorithm work (or not work, as the case may be) in the first place?

The answer is that step 5 must not be present, but it is critical that the array of items be passed by value as a parameter, and not be made a global variable.

Why does that make any difference, you ask? Consider passing the array as a parameter. If I "hire an assistant" and give that assistant a copy of my array, then the items in my own array are not rearranged as the assistant goes about his or her work. On the other hand, if the array is a global variable, then every time the assistant switches two items, those items are also switched in my version of the array.

You can see how this works by considering the case where N = 4. The first time the procedure is entered, L = 1. The array contains ABCD.

1) K = 1. I switch items 1 and 1 (accomplishing nothing). I then give an assistant a copy of the array with the instructions to find all permutations of the last 3 items. Notice that this will give all permutations of the four letters that start with A.
2) K = 2. I switch items 1 and 2, yielding BACD. I then hire an assistant to find all permutations of the last 3 items.
3) K = 3. I switch items 1 and 3, yielding CABD, and continue as before.
4) K = 4. I switch items 1 and 4, yielding DABC, and continue as before.

In other words, the algorithm successively moves each item to the front of the array and then calls itself recursively to find the permutations of the remaining letters. Everything is kept in the proper order, as long as "the assistant" doesn't rearrange my letters. If the array is made a global variable, then it's not immediately clear what happens, and it's interesting that the program could be more or less salvaged by step 5 above.

Moral of the story: Don't play around with a program you do not thoroughly understand. The change you make may be innocuous, but you are in no position to make that decision.

Math 287 Project 4 due Oct. 1, 1987
Multiplying matrices of complex numbers

In class we saw how to write a function that multiplies two matrices. We also saw how to represent complex numbers as records, as well as how to write a function that adds them. In this project, you are to combine those two features in a program that multiplies matrices of complex numbers.

The program should have a loop. Each time through the loop, the user has 4 options: enter a matrix, print a matrix, multiply two matrices, or quit. The matrices are to be indentified by numbers from 1 to 9. Allow them to have up to 15 rows or columns. I have attached output from my program. Your program must perform the exact same computations, except for the optional extra credit features at the end. Please turn in a listing of your program as well as the output, which should include the same computations I have included in my output.

Do not represent matrices exactly the way we did in class. Instead, represent them as records, where the entries are one part of the record, and the rows and columns are another.

For the weak at heart:

To keep the project from becoming too long, your program need not do any error checking, and will still receive full credit if it is well written and runs correctly when the input is correct.

For the masochistic:

There are gobs of features you could add to your program. In my output, you will note that I checked for matrices that cannot be multiplied together, matrices that do not exist and so cannot be printed or multiplied, invalid numbers of rows or columns, an invalid number for a matrix to be entered, printed, or multiplied, and an invalid choice from the options given. I should warn you that, with these options, my program was five pages long. You know from the last program what I can fit in a three page program. Allocate time carefully lest you end up with a monstrously long program that does everything but work correctly.

There is much more that could be done. For example, the program could read matrices from and write them to disk files. It could accept string input such as "Enter M1" or "M2=M1*M1". These types of features are often found in commercial programs.

Be sure to show me from your output any extra credit features you have done. In fact, highlight that part of the output so I don't miss it.

```
Program Project4 (input,output);
{------------------------------------------------------------------------}
{                                                                        }
{           Carolyn Walters                                              }
{           Math 287 - Project 4                                         }
{           Multiplying Matrices of Complex Numbers                      }
{           Sept. 17, 1987                                               }
{                                                                        }
{------------------------------------------------------------------------}
{ Program gives user choice to input a matrix, print a matrix, multiply  }
{ two matrices or to quit.  The elements of each matrix are complex      }
{ numbers (a + bi).  User is allowed to enter up to 9 different matrices.}
{ Each matrix may have up to 15 rows and columns.                        }
{------------------------------------------------------------------------}

Const  rowlim = 15;
       collim = 15;
       matlim = 9;

Type   comptype = record
                    reel,imag : real;
                  end;   { represents a complex number }

       matrixtype = array[1..rowlim,1..collim] of comptype;

       mattype  = record
                    row,col    : integer;
                    isamatrix  : Boolean;       { to tell if matrix is used }
                    matrix     : matrixtype;
                  end;  { mattype }

      arytype   = array[1..matlim] of mattype;     { array of matrices }

Procedure InitialMat (var matlist : arytype);
{ Empties the list of matrices. None exist at this point. }

var   i : integer;
begin
   for i := 1 to matlim do
       matlist[i].isamatrix := false;
end;
{-------------------- initialmat ------------------------------}
```

```
Procedure Menu (var choice:integer);
{ Lists choices and tests for a proper menu choice.}

begin
   choice := 0;
   While (choice < 1) or (choice > 4) do
   begin
      writeln;
      writeln('+++++++++++++++++++++++++++++++++++++++++++++++++');
      writeln('+                  MENU CHOICES                 +');
      writeln('+                                               +');
      writeln('+         1.  enter a matrix                    +');
      writeln('+         2.  print a matrix                    +');
      writeln('+         3.  multiply matrices                 +');
      writeln('+         4.  quit                              +');
      writeln('+                                               +');
      writeln('+++++++++++++++++++++++++++++++++++++++++++++++++');
      writeln;
      write('Please input a legal menu choice.  ');
      readln(choice);
      writeln;
   end;   { while }
end;  { menu }
{-------------------- menu ----------------------------------}

Procedure MultComplex (var c3:comptype; c1,c2:comptype);
{ Multiplies two complex numbers: (a+bi)*(c+di)= ac-bd + (bc+ad)i }

begin
   c3.real := c1.real * c2.real - c1.imag * c2.imag;
   c3.imag := c1.real * c2.imag + c1.imag * c2.real;
end;
{-------------------- multcomplex ---------------------------}

Procedure GetMatNum (var num : integer);
{ Gets a valid matrix number to be stored in list.}

begin
   num := 0;
   While (num < 1) or (num > matlim) do
   begin      { valid number }
      writeln;
      write('Enter a number 1 thru ',matlim:1,' to identify the matrix:  ');
      readln(num);
      writeln;
   end;       { valid number }
end;     { getmatnum }
{---------------------- getmatnum ----------------------------}
```

```
Procedure GetMatrix(var m : mattype);
{ Gets number of rows & columns for the matrix.  Tests for valid }
{ input for these.  Also gets the elements of the matrix.        }

var  r,c : integer;      { needed to get elements of matrix }

begin
   writeln;
   writeln('The limit for rows and columns of the matrix is ',rowlim:1,'.');
   write('Enter the number of rows for the matrix.  ');
   readln(m.row);
   writeln;

   While (m.row < 1) or (m.row > rowlim) do
   begin   { getting a valid row number }
      writeln('You have entered an invalid number. ');
      writeln('Number of rows must be from 1 to ',rowlim:1);
      write('Please enter a valid number.  ');
      readln(m.row);
      writeln;writeln;
   end;    { valid row }

   write('Enter the number of columns for the matrix.  ');
   readln(m.col);
   writeln;

   While (m.col < 1) or (m.col > collim) do
   begin   { getting valid column number }
      writeln('You have entered an invalid number.');
      writeln('Number of columns must be from 1 to ',collim:1);
      write('Please enter a valid number.  ');
      readln(m.col);
      writeln;writeln
   end;  { valid column }

   writeln('Enter each element as real part, space, imaginary part.');

   for r := 1 to m.row do
       for c := 1 to m.col do
       begin
          write('Enter element in row ',r:1,',column ',c:1,': ');
          readln(m.matrix[r,c].reel,m.matrix[r,c].imag);
          writeln;
       end;

   m.isamatrix := true;     { there is a matrix here now }
end;  { getmatrix }
{-------------------- getmatrix ------------------------------}
```

```
Procedure PrintMat (m:mattype;matnum:integer);
{ Prints out the matrix. Takes into account if imaginary part is }
{ negative.  Avoids +-2i.                                        }

var  r,c : integer;

begin
   writeln;
   writeln('  M A T R I X  ',matnum:1);
   writeln;
   for r := 1 to m.row do
   begin    { for r }
       for c := 1 to m.col do
           begin  { for c }
           write(m.matrix[r,c].reel:1:1);
           if m.matrix[r,c].imag  >= 0
              then write(' + ',abs(m.matrix[r,c].imag):1:1,'i   ')
              else write(' - ',abs(m.matrix[r,c].imag):1:1,'i   ');
           end;   { for c }
       writeln;
   end;     { for r }
end;  { printmat }
{------------------- printmat ---------------------------------}

Procedure MatMult (var m3:mattype; m1,m2 : mattype);
{ Multiplies two matrices and puts result in third. }

var   i,j,k  : integer;
      c : comptype;        { complex number resulting from 1 multiplication }

begin
   m3.row := m1.row;
   m3.col := m2.col;
   for i := 1 to m1.row do
       for j := 1 to m2.col do
       begin  { for j }
          m3.matrix[i,j].reel := 0;
          m3.matrix[i,j].imag := 0;
          for k := 1 to m1.col do
          begin   { for k }
             MultComplex(c,m1.matrix[i,k],m2.matrix[k,j]);
             m3.matrix[i,j].reel := m3.matrix[i,j].reel + c.reel;
             m3.matrix[i,j].imag := m3.matrix[i,j].imag + c.imag;
          end;  { for k }
       end;   { for j }
    m3.isamatrix := true;    { there is a matrix here }
end;  { matmult }
{------------------- matmult ---------------------------------}

Procedure TestNum(var num : integer);
{ Tests for a valid number of the matrix to be used. }

begin
   while (num < 1) or (num > matlim) do
   begin
      writeln('You have input an invalid matrix number.');
      write('Number must be from 1 to ',matlim:1,'.. Enter again. ');
      readln(num);
      writeln;
   end;
end;
{------------------- testnum ---------------------------------}

Procedure Multiply(var matlist : matrices;
```

```
Procedure Multiply(var matlist : arytype);
{ Asks for the matrices to be multiplied and determines if the operation }
{ is possible, whether the matrices exist or the columns of the first    }
{ are equal to the rows of the second.  If possible, it then calls the   }
{ procedure to do the actual multiplying.                                }

var first,second,third : integer; { subscripts of matrices }
    cando,candoit : Boolean;      { if multiplication can be done }

begin
   writeln;
   write('Enter number of the first matrix to be multiplied: ');
   readln(first);
   writeln;
   TestNum(first);
   writeln;
   write('Enter number of the second matrix to be multiplied: ');
   readln(second);
   TestNum(second);
   writeln;
   write('Enter number for the resulting matrix to be stored: ');
   readln(third);
   TestNum(third);
   writeln;

   cando := true;   { assumes there are matrices }
   if first = second
      then if not matlist[first].isamatrix
              then begin   { squaring a non-existent matrix }
                     write('Matrix ',first:1,' does not exit. ');
                     writeln('You cannot multiply.');
                     cando := false;
                   end     { squaring }
              else
       else begin { two distinct matrices }
             if not matlist[first].isamatrix
                then begin
                     write('Matrix ',first:1,' does not exist. ');
                     writeln('You cannot multiply.');
                     cando := false;
                     end;
             if not matlist[second].isamatrix
                then begin
                     write('Matrix ',second:1,' doesn't exist. ');
                     writeln('You cannot multiply.');
                     cando := false;
                     end;
          end;  { two distinct matrices }

   if cando then
   begin   { matrices exist }
   if matlist[first].col = matlist[second].row
      then candoit := true
      else candoit := false;
   if candoit then MatMult(matlist[third],matlist[first],matlist[second])
         else begin
              write('Matrix ',first:1,' has ',matlist[first].col:1);
              write(' columns and Matrix ',second:1,' has ');
              writeln(matlist[second].row:1,' rows.');
              writeln('Since they are not equal, they cannot be multiplied.
              end;
   end; { matrices exist }
end;  { multiply }
{-------------------- multiply --------------------------------}
```

```
Procedure Quit (var choice : integer);
{ Lets user decide if (s)he really wants to quit. }

var response : char;

begin
   writeln;
   write('You have decided to QUIT. Do you really want to QUIT? Y or N ');
   readln(response);
   writeln;
   While (response <> 'Y') and (response <> 'y') and (response <> 'N')
               and (response <> 'n') do
   begin
      write(' Please enter Y or N only: ');
      readln(response);
      writeln;
   end;
   if (response = 'Y') or (response ='y')
      then choice := 4
      else choice := 0;     { allows another menu choice }
end;  { quit }

Procedure Driver;
{ Controls flow of program. }

var   choice,i,matnum : integer;
      matlist  : arytype;

begin
   for i := 1 to matlim do InitialMat(matlist);
   choice := 0;
   while choice <> 4 do
   begin  { choice }
      menu(choice);
      case choice of
      1 : begin
            GetMatNum(matnum);
            GetMatrix(matlist[matnum]);
          end;  { case 1 }
      2 : begin
            GetMatNum(matnum);
            if matlist[matnum].isamatrix
                  then PrintMat(matlist[matnum],matnum)
                  else writeln('There is no matrix to be printed.');
          end;  { case 2 }
      3 : Multiply(matlist);
      4:  begin
           Quit(choice);
           if choice = 4 then
           begin  { if then }
             writeln;
             writeln('Thanks for running this program.  Have a nice day!!');
             writeln('G O O D B Y E !!!!');
           end;   { if then }
          end;  { case 4 }
      end;  { case choice }
   end;   { while }
end;
{------------------- driver ---------------------------------}

begin  { main }
   driver;
end.   { main }
```

```
{$P128,G128}
PROGRAM PROJECT3 (INPUT, OUTPUT);
{Frances Marcello            October 1, 1987       Math 287 }

{This program will present a menu with the following options-
          1. Enter a matrix
          2. Print a matrix
          3. Multiply two matrices
          4. Quit                                              }


TYPE
    COMPLEX      = RECORD
                     REEL,           {Real part of complex number}
                     IMAG:REAL;      {Imaginary part of complex number}
                   END;
    MATRIXDATA   = ARRAY [1..15,1..15] OF COMPLEX;
    MATRIXTYPE   = RECORD
                     FULL:BOOLEAN;   {Indicate status of matrix}
                     NR,             {Number of rows in matrix}
                     NC:INTEGER;     {Number of columns in matrix}
                     DATA:MATRIXDATA;{Complex numbers in matrix}
                   END;
    MATRIX       = ARRAY[1..9] OF MATRIXTYPE;


VAR
    Code          :INTEGER;        {Selection code for menu}
    MenuChoice    :INTEGER;
    M             :MATRIX;         {Matrix}
    CR            :INTEGER;        {Count of Rows}
    CC            :INTEGER;        {Count of Columns}

{*************************************************************************}

PROCEDURE INITIALIZEARRAYS;
{This procedure will initialize all arrays at zero}

VAR
    Number:INTEGER;      {Number - the subscript for the arrays}

    BEGIN  {Procedure INITIALIZEARRAYS}
      FOR Number := 1 TO 9 DO
        BEGIN  {FOR Number}
          WITH M [Number] DO
            BEGIN  {WITH M}
              FULL:=FALSE;
              NR:=0;
              NC:=0;
              FOR CR := 1 TO 15 DO
                FOR CC := 1 TO 15 DO
                  BEGIN  {FOR CC}
                    DATA[CR,CC].REEL := 0.0;
                    DATA[CR,CC].IMAG :=0.0;
                  END;   {FOR CC}
```

```
          END;   {WITH M}
      END;   {FOR Number}
  END;   {Procedure INITIALIZEARRAYS}

{*************************************************************************}

PROCEDURE CHECKCODE (Min,Max:INTEGER; VAR Code:INTEGER);
{This procedure will check for acceptable code numbers as designated.
 Code number must be between a minimum value (Min) and a maximum value
 (Max) inclusive.}

  BEGIN   {Procedure CHECKCODE}
    IF (Code >= Min) AND (Code <= Max)
    THEN {Acceptable response}
    ELSE
      BEGIN  {Unacceptable response}
        WRITELN;
        WRITELN ('          * * * * * The entry of ',
                 Code:1, ' is invalid * * * * * ');
        WRITELN;
        WRITELN ('              PLEASE REENTER  :',Min,
                   ' <= ENTRY <=', Max);
        WRITE ('             ENTRY:');
        READLN (Code);
        CHECKCODE (Min,Max,Code);
    END; {Unacceptable response}
  END;   {Procedure CHECKCODE}

{*************************************************************************}

PROCEDURE MENU   (VAR  Code:INTEGER);
{This procedure will display the program menu and get the user selection}

  BEGIN  {Procedure MENU}
    CLRSCR;
    WRITELN;
    WRITELN;
    WRITELN;
    WRITELN ('            MATRIX ARITHMETIC     ');
    WRITELN;
    WRITELN;
    WRITELN ('    ---------------------------------------');
    WRITELN ('                  MENU');
    WRITELN ('    ---------------------------------------');
    WRITELN;
    WRITELN;
    WRITELN ('       CODE            FUNCTION');
    WRITELN;
    WRITELN ('         1   . . . .  ENTER A MATRIX');
    WRITELN ('         2   . . . .  PRINT A MATRIX');
    WRITELN ('         3   . . . .  MULTIPLY TWO MATRICES');
    WRITELN ('         4   . . . .  EXIT FROM PROGRAM');
    WRITELN;
    WRITELN;
    WRITE ('         ENTER CODE 1, 2, 3, OR 4:');
```

```
      READLN (Code);
      CHECKCODE (1,4,Code);
    END;  {Procedure MENU}

{*************************************************************************}

PROCEDURE ENTERMATRIX;
{This procedure will allow entry of a ordered matrix of complex numbers}

VAR

    I      :INTEGER;   {I- the subscript index of the matrix}
    C1,C2  :INTEGER;   {C1,C2- counters in FOR loops}

    BEGIN  {Procedure ENTERMATRIX}
      CLRSCR;
      WRITELN;
      WRITELN(' The matrix to be read in is to be numbered 1 to 9');
      WRITE(' Please enter the appropriate identification number: ');
      READLN (I);
      CHECKCODE (1,9,I);
      WRITELN;WRITELN;
      WITH M [I] DO
        BEGIN  {WITH M}
          WRITELN('* * * The maximum matrix size is 15 rows X 15 columns * * *')
;
          WRITE(' Enter the number of rows for MATRIX ',I,':  ');
          READLN (NR);
          CHECKCODE(1,15,NR);
          WRITELN;
          WRITE(' Enter the number of columns for MATRIX ',I,':  ');
          READLN(NC);
          CHECKCODE(1,15,NC);
          WRITELN;
          WRITELN('Enter complex numbers according to the following format:');
          WRITELN('                                                    REAL SPACE IMAGI
NARY ');
          FOR C1:=1 TO NR DO
            FOR C2 := 1 TO NC DO
              BEGIN  {FOR C2}
                WRITE('Enter complex number in row ',C1:2,
                                  ' column ',C2:2, ':   ');
                WITH DATA[C1,C2] DO
                   READLN(REEL,IMAG);
              END;   {FOR C2}
        FULL:=TRUE
        END;   {WITH M}
    END;   {Procedure ENTERMATRIX}

{*************************************************************************}

PROCEDURE PRINTMATRIX;
{This procedure will allow the printing of a ordered matrix of complex numbers}

VAR
```

```
    I    :INTEGER;  {I - the subscript index of the matrix}
    C1,C2:INTEGER;  {C1,C2- counters in FOR loops}
    Dummy:CHAR;     {Dummy- Input variable to continue}

    BEGIN  {Procedure PRINTMATRIX}
      CLRSCR;
      WRITELN;
      WRITELN(' The matrix to be printed is to be numbered 1 to 9');
      WRITE(' Please enter the appropriate identification number: ');
      READLN (I);
      CHECKCODE (1,9,I);
      WRITELN;
      WITH M [I] DO
        BEGIN   {WITH M}
          IF FULL
            THEN
              BEGIN  {THEN}
                WRITELN(' MATRIX ',I:2,'HAS ',NR:3,' ROWS AND ',
                                                NC:3,' COLUMNS');
                WRITELN;
                FOR C1:=1 TO NR DO
                  BEGIN  {FOR C1}
                    FOR C2 := 1 TO NC DO
                       BEGIN  {FOR C2}
                         WITH DATA[C1,C2] DO
                              IF IMAG >= 0.0
                              THEN
                                WRITE(REEL:15:2, '+',IMAG:5:2 ,'I     ')
                              ELSE
                                WRITE(REEL:15:2,'-',-IMAG:5:2,'I     ');
                       END;   {FOR C2}
                    WRITELN;
                  END;   {FOR C1}
                WRITELN;
                WRITE('PRESS RETURN TO CONTINUE:');
                READLN(DUMMY);
              END   {THEN}
            ELSE
              BEGIN  {ELSE}
                WRITELN(' MATRIX ',I:3,' IS EMPTY AS YET');
                WRITE('PRESS RETURN TO CONTINUE');
                READLN(Dummy);
              END;  {ELSE}
        END;    {WITH M}
    END;   {Procedure PRINTMATRIX}

{*************************************************************************}

PROCEDURE MULTIPLYMATRICES;
{This procedure will multiply two matrices when possible}

CONST
    Message = ' These matrices cannot be multiplied ';

VAR
```

```
    C1,C2,C3     :INTEGER;
    I1,I2,I3     :INTEGER;
    A,B,C,D      :REAL;
    Dummy        :CHAR;

  BEGIN  {Procedure MULTIPLYMATRICES}
    CLRSCR;
    WRITELN (' USE IDENTIFICATION NUMBERS 1 TO 9 ');
    WRITE(' Enter the identification number of the first matrix: ');
    READLN(I1);
    CHECKCODE(1,9,I1);
    WRITE(' Enter the identification number of the second matrix: ');
    READLN(I2);
    CHECKCODE(1,9,I2);
    WRITE(' Enter the identification number for the product matrix: ');
    READLN(I3);
    CHECKCODE(1,9,I3);
    IF M [I1].NC <> M [I2].NR
       THEN
         BEGIN {THEN}
           WRITELN(MESSAGE);
           WRITELN('PRESS RETURN TO CONTINUE');
           READLN(DUMMY);
         END {THEN}
       ELSE
         BEGIN   {ELSE}
           M [I3].NR:= M [I1].NR;
           M [I3].NC:= M [I2].NC;
           FOR C1 :=1 to M [I1].NR DO
             FOR C2 := 1 TO M [I2].NC DO
               BEGIN  {FOR C2}
                 M [I3].DATA [C1,C2].REEL:=0.0;
                 M [I3].DATA [C1,C2].IMAG:=0.0;
                    FOR C3 := 1 TO M [I1].NC DO
                      BEGIN  {FOR C3}
                        A:=M [I1].DATA [C1,C3].REEL;
                        B:=M [I1].DATA [C1,C3].IMAG;
                        C:=M [I2].DATA [C3,C2].REEL;
                        D:=M [I2].DATA [C3,C2].IMAG;
                        WITH M [I3].DATA [C1,C2] DO
                          BEGIN  {WITH M}
                            REEL:=REEL + (A*C - B*D);
                            IMAG:=IMAG + (A*D + B*C);
                          END;        {WITH M}
                        M [I3].FULL:=TRUE
                      END;   {FOR C3}
               END;        {FOR C2}
         END;    {ELSE}
  END;        {Procedure MULTIPLYMATRICES}

{*****************************************************************************}

  BEGIN  {Main program}
    MenuChoice := 0;
```

```
  INITIALIZEARRAYS;
  REPEAT
  WRITELN;WRITELN;
  WHILE MenuChoice <> 4 DO
    BEGIN  {While}
      MENU(MenuChoice);
      CASE MenuChoice OF
       1: ENTERMATRIX;
       2: PRINTMATRIX;
       3: MULTIPLYMATRICES;
      END;   {CASE MenuChoice}
    END; {While}
  WRITELN (' END OF PROGRAM');
  UNTIL MenuChoice = 4;
END.   {Main program}
```

Math 299A Project 3
Due Tuesday, July 26

If it's good enough for Educational Testing Service, it's good enough for us (sometimes).

This project is based on question 3, section II, of the 1986 Advanced Placement Examination in Computer Science. A copy of that question is attached; please read it before going on.

You're back? Okay. Part of this project is to write the procedure requested in question 3. Before doing that, you first need to create a tree. This may be done using the procedure Insert on p. 405 of Dale and Lilly. But to receive full credit for this project, you must instead use a recursive procedure to accomplish the same thing. Don't panic; it's easier to write than the nonrecursive version in the text. After the tree is constructed, use the recursive version of InOrder on p. 421 to print the contents of the tree in order.

Once you get that working, then write the procedure NumberLeaves. Your program should ask for a value of NextNum to be entered, and then call NumberLeaves. Finally, the leaves of tree should be printed in order, as well as the final value of NumberLeaves. For example, if the tree in question 3 were used, the last part of the output would be:

```
6
5
4
3
The final value of NextNum is 7.
```

Run your program using the sample tree given in question 3. It should not be hard to come up with some numbers that would cause the sample tree to be constructed. On your output, draw a picture of what the tree should look like using the numbers you entered.

When entering the numbers to be put into the tree, let a negative number signal the end of the input.

I can't think of any extra credit options; can you?

3. In a binary tree, a leaf x is said to be to the left of a leaf y if there exists a node n in the tree such that x is in the left subtree of n and y is in the right subtree of n.

The following declarations are used for a binary tree.

```
type
  TreeType = ↑Node ;
  Node = record
           Datum : integer ;
           Left, Right : TreeType
         end ;
```

Write a procedure with the heading:

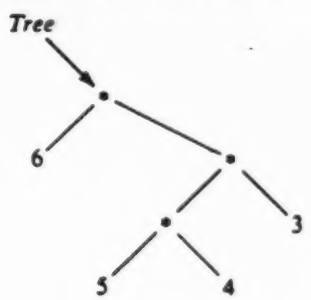
```
procedure NumberLeaves(Tree : TreeType ; var NextNum : integer) ;
```

The procedure should assign numbers to the Datum fields of the leaves in such a way that the leaves are numbered consecutively from right to left. The numbering should start with the value that the var parameter *NextNum* has when the procedure is called. When the procedure terminates, *NextNum* should be 1 greater than the number assigned to the leftmost leaf. The Datum fields of nodes that are not leaves should not be modified. (You may assume that any and all nodes of the tree pointed to by *Tree* have been properly initialized.)

For example, the effect of

```
NextNum := 3 ;
NumberLeaves(Tree, NextNum)
```

is to number the leaves of *Tree* as follows



and to set *NextNum* to 7.

```
Program project3 (input,output,filel);
{++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++}
{                                                                      }
{                    Carolyn Walters                                   }
{                    July 17, 1988                                     }
{                    Project 3 - Trees                                 }
{                                                                      }
{++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++}
{                                                                      }
{ This program creates a tree of integers.  The leaves of the          }
{ tree are then marked from some starting value in the order           }
{ right to left.  I have allowed the user to try each                  }
{ procedure to make the tree, one done recursively and the             }
{ other non-recursively.  Input can be from the keyboard or a          }
{ file on disk.  User can enter other trees.  I have checked           }
{ for empty tree and have a procedure to clear the previous            }
{ tree.   The tree will be printed and then the tree with the          }
{ leaves marked will also be printed before just printing the          }
{ leaves.                                                              }
{                                                                      }
{++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++}

type    infotype = integer;                    { data in treenode }
        treetype = ^treenode;

        treenode = record
                     datum : infotype;
                     left,right : treetype;
                   end;


Procedure Directions;
begin
   writeln;
   writeln('This program will use a tree data structure.');
   writeln('Positive integers will be put on the tree and ');
   writeln('the leaves of this tree will be marked from ');
   writeln('right to left.  The value of these marked leaves');
   writeln('will be printed.');
   writeln;
end;
{--------------------  end directions  ----------------------}

Procedure Menu(var choice : char);
begin
    writeln('+++++++++++++++++++++++++++++++++++++++');
    writeln('+                                     +');
    writeln('+            M E N U                  +');
    writeln('+                                     +');
    writeln('+  1. make tree non-recursively       +');
    writeln('+  2. make tree recursively           +');
    writeln('+                                     +');
    writeln('+++++++++++++++++++++++++++++++++++++++');
    writeln;
    repeat
        write ('Enter your valid choice here ----> ');
        readln(choice);
        writeln;
    until choice in ['1','2'];
    writeln;
 end;  { menul }
{--------------------  end menul ----------------------------}
```

```
Procedure Menu2(var choice2 : char);
begin
   writeln;
   writeln('****************************************');
   writeln('*                                      *');
   writeln('*            M E N U                   *');
   writeln('*                                      *');
   writeln('*   1. enter data from keyboard        *');
   writeln('*   2. enter data from file on disk    *');
   writeln('*                                      *');
   writeln('****************************************');
   repeat
      writeln;
      write('Enter your valid choice here ----> ');
      readln(choice2);
      writeln;
   until choice2 in ['1','2'];
end;   { menu2 }
{----------------------  end menu2 ----------------------}

Procedure MakeTree (var tree : treetype; value : infotype);
{ this is the non-recursive method of making a tree }
var   p,q,temp : treetype;

begin
   new(temp);                              { setting up new node }
   temp^.datum := value;
   temp^.left := nil;
   temp^.right := nil;

   p := tree;
   q := nil;

   while p <> nil do                       { finding place to put on tree }
   begin
        q := p;
        if p^.datum > value
           then p := p^.left
           else p := p^.right;
  end;   { p <> nil }

  if q = nil                               { putting on tree }
     then tree := temp
     else if q^.datum > value
             then q^.left := temp
             else q^.right := temp;

end;   { make tree }
{----------------------  end maketree ----------------------}
```

```
Procedure ReMakeTree (var tree : treetype; value : infotype);
{ this is recursive method of making a tree }
var   temp : treetype;
begin

       if tree <> nil
          then    if tree^.datum > value
                   then ReMakeTree(tree^.left,value)
                   else ReMakeTree(tree^.right,value)
           else begin
                   new (temp);                  { setting up new node }
                   temp^.datum := value;        { and putting on tree }
                   temp^.left := nil;
                   temp^.right := nil;
                   tree := temp;
                end;
end;    { remake }
{----------------------  end remaketree  ----------------------}

Procedure PrintInOrder (tree : treetype);
{ prints tree in order }
begin
     if tree <> nil
        then  begin
                   PrintInOrder(tree^.left);
                   writeln(tree^.datum:5);
                   PrintInOrder(tree^.right);
              end;

end;   { in order }
{--------------------  end printinorder --------------------}

Procedure PrintLeaves(tree : treetype);
{ prints only the leaves of tree in order }
begin
     if tree <> nil
        then begin
               PrintLeaves(tree^.left);
               if (tree^.left = nil) and (tree^.right = nil)
                  then writeln(tree^.datum:5);
               PrintLeave(tree^.right);
             end;
end;   { printleaves }
{--------------------  end printleaves  ----------------------}

Procedure NumberLeaves(tree : treetype; var nextnum : infotype);
{ replaces value in a leaf with the value of nextnum }
{ starts at the right and works left                 }
begin
     if (tree^.left = nil) and (tree^.right = nil)
        then begin
                   tree^.datum := nextnum;
                   nextnum := nextnum + 1;
             end
        else begin
                  if tree^.right <> nil
                     then NumberLeaves(tree^.right,nextnum);
                  if tree^.left <> nil
                     then NumberLeaves(tree^.left,nextnum);
             end;
end;    { numberleaves }
{----------------------  end numberleaves --------------------}
```

```
Procedure GetRidTree(var tree : treetype);
{ clears the nodes of the tree }
begin
    if (tree^.left = nil) and (tree^.right = nil)
       then dispose(tree)
       else begin
               if tree^.left <> nil
                  then GetRidTree(tree^.left);
               if tree^.right <> nil
                  then GetRidTree(tree^.right);
            end;
end;    { getridtree }
{--------------------  end getridtree  ----------------------}

Procedure GetData (var infile : text; var tree : treetype; flag1,flag2 : char
{ gets the integers for the tree using flag2 to get input  }
{ from keyboad or file and flag1 to make the tree by using }
{ a recursive or non-recursive procedure                   }

var value : infotype;
begin
    repeat
       if flag2 = '1'
          then begin
                  writeln;
                  writeln('A NEGATIVE NUMBER WILL STOP INPUT.');
                  writeln;
                  write('Enter an integer here ----> ');
               end;
       readln(infile,value);
       if flag2 = '1' then writeln;writeln;
       if flag1 = '1'
          then if value > 0
                  then MakeTree(tree,value);
       if flag1 = '2'
          then if value > 0
                  then ReMakeTree(tree,value);
    until value < 0;
end;    { getdata }
{--------------------  end getdata --------------------------}
```

```
Procedure DoLeaves(var tree : treetype);
{ asks for starting number to mark leaves and prints both the }
{ entire marked tree and then just the leaves                  }

var  nextnum : integer;
begin
    write('Enter the number to start marking the leaves. ----> ');
    readln(nextnum);
    writeln;writeln;
    NumberLeaves(tree,nextnum);
    writeln;
    writeln('Here are the values after marking the leaves.');
    writeln;
    PrintInOrder(tree);
    writeln;
    write('Press enter to continue ---->');      { for screen control }
    readln;
    writeln;
    writeln('Here are the numbered leaves:');
    writeln;
    PrintLeaves(tree);
    writeln;
    writeln('The final value of the nextnum is ',nextnum:1);
    writeln;
end;  { doleaves }
{-----------------------  end doleaves  ----------------------}
```

```
Procedure Driver;
{ controls flow of program, initializes tree and checks of }
{ empty tree and files, allows user to enter other trees   }

var   tree    : treetype;
      choice  : char;              { recursive or non-recursive }
      choice2 : char;              { keyboard or file input     }
      another : char;              { for additional trees       }
      filel   : text;
begin
     Directions;
     repeat
        tree := nil;
        Menu(choice);
        Menu2(choice2);
        If choice2 ='1'
           then GetData(input,tree,choice,choice2)
           else begin
                   write('Enter file name here ---> ');
                   reset(filel);
                   if eof(filel)
                      then writeln('THIS FILE IS EMPTY.')
                      else GetData(filel,tree,choice,choice2);
                   close(filel);
                end;
        writeln;

        if tree = nil
           then writeln('THIS TREE IS EMPTY.')
           else begin
                  writeln('Here is the tree in order: ');
                  writeln;
                  PrintInOrder(tree);
                  write('Press enter to continue ---->');
                  readln;
                  writeln;writeln;
                  DoLeaves(tree);
                end;    { else not empty }

        repeat
            writeln;
            write('Do you wish to enter another tree?  Y or N ---> ');
            readln(another);
            writeln;
        until another in ['Y','N','y','n'];     { test input }

        if tree <> nil
        then repeat
                GetRidTree(tree);
             until tree = nil;

     until another in ['N','n'];                { end run }
end;    { driver }


begin
   Driver;
end.
```

Math 299 Project 2
Due July 19, 1988

In this project you are to write a program that enters names from the keyboard and puts them in alphabetical order using a linked list. This much of the project should be straightforward, since the insertElement procedure is already given to you in the book. Suppose this list represents a class list. Your program should let you later decide to split the class by taking all names between certain letters and putting them in a second list. For example, if you use the TTI roster for this summer and entered 'fm', all the names between Fitzgerald and Mulz would be put into a second list.

Run the program with the TTI roster, taking every other name for simplicity. Also, for simplicity, enter just the last names and use all lower case letters. Enter the names in a random order, with the program inserting each new name in the appropriate spot in the linked list. Then print the list to ensure that the names were entered correctly. (Hint: you may want to get this part of the program working before trying the next part.) Then try entering 'fm'. Have the program print out both lists. Once this is working, try other pairs of letters, such as 'qq', 'ap', 'lz', 'az', and 'zz'. (Another hint: although you don't need the DeleteElement procedure for this project, there are parts of this procedure you may find useful for picking out a subset of the original list.)

One other twist: your linked lists should have a header node that keeps track of how many elements are in the linked list. Notice that the header node has an integer in its information part, while the other nodes have a string there.

As usual, your program should be well written, showing proper use of procedures, functions, comments, and data structures.

There are tons of things you can try for extra credit:

--Allow your program to handle upper and lower case letters, so d'Amato would be in between Casey and Dole.

--Allow your program to handle incorrect input, such as 'pa' or '$t', or with blanks in the input, such as 'f m'.

--Allow your program to ask for more than one pair of letters, and put all those subsets in the appropriate places in the second list. Better still, have a whole bunch of linked lists, and allow the program to take a subset of any list and put that subset in the appropriate place in any other list. The program could continue splitting and combining lists, and printing whatever lists are requested, until the user decides to quit.

Be sure to highlight any extra credit work on your output so I can give you credit for it.

```
File = PROJ24/PCL        LRL = 256        REC = ASCII
Program Project2  (input,output,filel);
{-------------------------------------------------------------}
{                                                             }
{                  Carolyn Walters                            }
{                  July 15, 1988                              }
{                  Project 2 - Linked Lists                   }
{                                                             }
{-------------------------------------------------------------}
{                                                             }
{ This program allows the user to enter a list of names from  }
{ the keyboard or a file on disk.  The names are put into     }
{ alphabetical order using linked lists and pointers.  The    }
{ names can be entered with capital or small letters.  It     }
{ does not deal with exceptions like McDonald or d'Amato, so  }
{ these type names are deliberately not used as data.  It     }
{ allows the user to create a sublist given certain limits    }
{ of the alphabet.  I have also included adding addition      }
{ names to the original list.  The number of names on the     }
{ list is printed.  The user can rejoin the sublist to the    }
{ original list and automatically does so when user wants to  }
{ make a different sublist.  Various testing is done - see    }
{ output with notes.  When user has finished with first list  }
{ a different list can be entered.  If you knew how slow my   }
{ compiler for the TRASH-80 is, you would understand why I    }
{ have not done more - like using an array for many sublists  }
{ and taking care of those oddball names.                     }
{                                                             }
{-------------------------------------------------------------}

const   maxlen = 20;                { length of name or string }
type    strtype = record
                  len : 0..maxlen;
                  str : array[1..maxlen] of char;
                  end;   { record }

        nametype = record
                     first, last : strtype;
                   end;

        nodeptr =  ^node;
        node    = record
                    info : nametype;
                    next : nodeptr;
                  end;   { record }

        headptr = ^headnode;
        headnode =  record
                      count : integer;
                      ptr   : nodeptr;
                    end;   { record }
        charset = set of char;

Procedure Directions;
begin
   writeln('This program will allow the user to make several choices.');
   writeln('You can enter a list from the keyboard or a file.');
   writeln('You can enter additional names and can break up the');
   writeln('original list into a sublist.');
   writeln;
end;
{---------------------- end directions -----------------------}
```

```
Procedure Menu1(var choice : char);
begin
   writeln('************************************************');
   writeln('*                                              *');
   writeln('*                  M E N U                     *');
   writeln('*                                              *');
   writeln('*    1.  Create list from keyboard             *');
   writeln('*    2.  Get list from file on disk            *');
   writeln('*                                              *');
   writeln('************************************************');
   writeln;
   repeat
      write('  Enter your choice ---> ');
      readln(choice);
      writeln;
   until (choice = '1') or (choice = '2');
end;
{---------------------- end menu1 ---------------------------}

Procedure Menu2(var choice : char);
begin
   writeln;
   writeln('++++++++++++++++++++++++++++++++++++++++++++++++');
   writeln('+                                              +');
   writeln('+                M E N U                       +');
   writeln('+                                              +');
   writeln('+      1.  Add new name to list                +');
   writeln('+      2.  Print the list                      +');
   writeln('+      3.  Create a new sublist                +');
   writeln('+      4.  Rejoin the two lists                +');
   writeln('+      5.  Quit this list                      +');
   writeln('+                                              +');
   writeln('++++++++++++++++++++++++++++++++++++++++++++++++');
   writeln;
   repeat
      write('Enter your choice here ----> ');
      readln(choice);
      writeln;
   until  choice in ['1','2','3','4','5'];
end;
{---------------------- end menu2 ---------------------------}

Procedure InitStr (var s:strtype);
{clears string array }
var   i : integer;
begin
  for i := 1 to maxlen do
      s.str[i] := ' ';
  s.len := 0;
end;
{---------------------- end initstr ---------------------------}
```

```
Procedure GetName (var infile : text; var name : nametype);
{ gets both first and last names }

var   i  : integer;
      ch : char;
begin
   InitStr(name.first);
   InitStr(name.last);
   ch := '?';
   i := 0;
   repeat
      i := i + 1;
      read(infile,ch);
      name.first.str[i] := ch;
   until (ch = ' ') or (i >= 20);
   name.first.len := i - 1;
   i := 0;
   repeat
      i := i + 1;
      read(infile,ch);
      name.last.str[i] := CH;
   until eoln (infile);
   name.last.len := i;
   if eoln(infile) then readln(infile);

end;   { getname }
{---------------------- end getname --------------------------}

Procedure ConvertStr (var s : strtype);
{ makes first letter a capital and all others small letters }
var  capset, smallset : charset;
     i : integer;
     ch : char;
begin
    capset := ['A'..'Z'];
    smallset := ['a'..'z'];
    if s.str[1] in smallset
       then s.str[1] := chr(ord(s.str[1]) - 32);
    for i := 2 to s.len do
        if s.str[i] in capset
           then s.str[i] := chr(ord(s.str[i]) + 32);
end;
{--------------------- end convertstr -------------------------}

Procedure ConvertName (var name : nametype);
begin
    ConvertStr(name.first);
    ConvertStr(name.last);
end;
{-------------------- end convertname -------------------------}
```

```
Procedure PrintName (var outfile : text; name : nametype);
var   i : integer;
begin
   for i := 1 to name.first.len do
       write(outfile,name.first.str[i]);
   write(outfile,' ');
   for i := 1 to name.last.len do
       write(outfile,name.last.str[i]);
   writeln(outfile);
end;
{------------------ end printname ----------------------------}

Procedure InitHead(var head : headptr);
begin
   new(head);
   head^.count := 0;
   head^.ptr := nil;
end;
{------------------ end inithead -----------------------------}

Procedure PutOnList( head:headptr;  name : nametype);
{ puts name on list in alphabetical order }
{ uses two pointers to mark place for insertion }

var   p,q : nodeptr;
      temp : nodeptr;
      found : Boolean;
begin
    new(temp);
    temp^.info := name;
    temp^.next := nil;
    p := head^.ptr;
    q := nil;
    found := false;
    while (p <> nil) and not found do
       if p^.info.last.str > name.last.str
          then found := true
          else if (p^.info.last.str = name.last.str) and
                  (p^.info.first.str > name.first.str)
                  then found := true
                  else begin
                          q := p;
                          p := p^.next;
                       end;
    temp^.next := p;
    If q = nil
       then head^.ptr := temp
       else  q^.next := temp;
    head^.count := head^.count + 1;
end;
{---------------------- end putonlist ------------------------}
```

```
Procedure PrintList (var outfile : text; var head : headptr);
var   p : nodeptr;
begin
     p := head^.ptr;
     if p = nil
        then writeln('List is empty. ')
        else begin
               writeln;
               writeln('The number of names on the list is ',head^.count:3,'.');
               writeln;
               writeln(' H E R E   I S   T H E   L I S T ');
               writeln;
               while p <> nil do
               begin
                  PrintName(outfile,p^.info);
                  p := p^.next;
               end;   { while p }
               writeln;
            end;   { else }
end;
{---------------------- end printlist --------------------------}

Procedure GetChar(var ch1,ch2 : char);
{ gets the limit letters for the sublist - tests for bad input }

var   capset,smallset : charset;
      dataok : Boolean;
begin
    dataok := false;
    capset := ['A'..'Z'];
    smallset := ['a'..'z'];
    repeat
       write('Enter Two valid letters ---> ');
       readln(ch1,ch2);
       If ch1 in smallset
          then ch1 := chr(ord(ch1) - 32);
       If ch2 in smallset
          then ch2 := chr(ord(ch2) - 32);
       If (ch1 in capset) and (ch2 in capset) and (ch1 <= ch2)
          then dataok := true
          else writeln('Entry not valid - try again. ');
    until dataok;
end;
{-------------------- end getchar ---------------------------}
```

```
Procedure GetNewList (head : headptr; var newhead : headptr; ch1,ch2 : char);
{ creates new sublist - checks for beginning of list and also }
{ if no names exist for the sublist }
{ finds the starting and stopping nodes and connects correct }
{ pointers }

var   start,stop,temp1,temp2 : nodeptr;   { pointers for linking lists }
      ch : char;
      findstart,findstop : Boolean;
      counter : integer;
begin
   newhead := nil;
   InitHead(newhead);
   start := head^.ptr;
   temp1 := nil;
   temp2 := nil;
   findstart := false;
   if start = nil
      then writeln(' List is empty. Why not try some other choice???')
      else begin   { else not nil }
             while (start <> nil) and not findstart do
             begin   { while start }
               ch := start^.info.last.str[1];
               if ch >= ch1
                  then findstart := true
                  else begin
                         temp1 := start;
                         start := start^.next;
                       end;
             end;    { while start }
             if findstart
             then begin   { if findstart }
                    counter := 0;
                    stop := start;
                    temp2 := temp1;
                    findstop := false;
                    while (stop <> nil) and not findstop do
                    begin    { while stop }
                       ch := stop^.info.last.str[1];
                       if ch > ch2
                          then findstop := true
                          else begin
                                 temp2 := stop;
                                 counter := counter + 1;
                                 stop := stop^.next;
                               end;
                    end;     { while stop }
                    if start = stop
                       then writeln('No names to put on new list.')
                       else begin
                              newhead^.count := counter;
                              newhead^.ptr := start;
                              if temp1 = nil
                                 then head^.ptr := stop
                                 else temp1^.next := stop;
                              temp2^.next := nil;
                              head^.count := head^.count - counter;
                            end;  { else new list }
                  end;   { findstart }
           end;   { else not nil }
end;   { procedure }
{----------------------- end getnewlist ----------------------}
```

```
Procedure GetRidList (var header : headptr);
var   p : nodeptr;
begin
     p := header^.ptr;
     while p <> nil do
     begin
       header^.ptr := p^.next;
       dispose(p);
       p := header^.ptr;
     end;
     header^.ptr := nil;
     header^.count := 0;
end;
{---------------------- end getridlist ----------------------}

Procedure RejoinList (var head,newhead:headptr);
{ joins the sublist back in correct place in original list }

var    p,q          : nodeptr;
       found        : Boolean;
begin
   if newhead^.ptr = nil
      then writeln (' Sublist is empty - nothing to rejoin. ')
      else begin
             p := head^.ptr;
             q := nil;
             found := false;
             while (p <> nil) and not found do
               if p^.info.last.str > newhead^.ptr^.info.last.str
                  then found := true
                  else begin
                         q := p;
                         p := p^.next;
                       end;
             if q = nil
                then head^.ptr := newhead^.ptr
                else q^.next := newhead^.ptr;

             q := newhead^.ptr;
             while q^.next <> nil do
                   q := q^.next;
             q^.next := p;
             head^.count := head^.count + newhead^.count;
             InitHead(newhead);
           end;    { else }
end;  { rejoin }
{--------------------- end rejoinlist ------------------------}

Procedure WhichList (var choice : char);
begin
   writeln (' 1.  To print first list. ');
   writeln (' 2.  To print sublist. ');
   repeat
       write(' Enter choice here ---> ');
       readln(choice);
   until choice in ['1','2'];
end;
{-------------------- end whichlist -------------------------}
```

```
Procedure Options(head : headptr);
{ gives user choices to perform with the original and new lists }

var   choice1,choice2 : char;
      ch1,ch2 : char;
      name : nametype;
      newhead : headptr;

begin
   InitHead(newhead);
   repeat
     Menu2(choice1);
     case choice1 of
      '1' : begin
              write(' Enter name here ---> ');
              GetName(input,name);
              writeln;
              ConvertName(name);
              PutOnList(head,name);
            end;   { case 1 }
      '2' : begin
              WhichList(choice2);
              if choice2 = '1'
                 then PrintList(output,head)
                 else PrintList(output,newhead);
            end;   { case 2 }
      '3' : begin
              if newhead^.ptr <> nil
                 then RejoinList(head,newhead);
              writeln('To get a sublist you must enter the ');
              writeln('first and last letters to create the');
              writeln('new list.  For example: ad would list');
              writeln('all names starting with an a to all ');
              writeln('names starting with d.');
              writeln;
              GetChar(ch1,ch2);
              GetNewList(head,newhead,ch1,ch2);
            end;   { case 3 }
      '4' : if newhead^.ptr <> nil
               then RejoinList(head,newhead)
               else writeln('Sublist is empty - try another option.');
      '5' : writeln(' Your choice to end this segment of the program. ');
       end;  { case choice }
   until choice1 = '5';
   if newhead^.ptr <> nil
      then GetRidList(newhead);
end;
{---------------------- end options -------------------------}
```

```
Procedure  Driver;
var   name     : nametype;
      response : char;              { controls input of names from keyboard }
      another  : char;              { allows different lists }
      head     : headptr;
      filel    : text;
      choice   : char;              { data choice from keyboard or file }
begin
      Directions;
      repeat
         InitHead(head);
         Menu1(choice);
         if choice = '1'
            then repeat
                    write('Enter name here ---> ');
                    GetName(input,name);
                    writeln;
                    ConvertName(name);
                    PutOnList(head,name);
                    write('Do you want to enter another name?  Y or N ---> ');
                    readln(response);
                    writeln;
                 until (response = 'N') or (response ='n')
              else begin
                    write('Enter file name ---> ');
                    reset (filel);
                    while not eof(filel) do
                    begin
                       GetName(filel,name);
                       ConvertName(name);
                       PutOnList(head,name);
                    end;    { not eof }
                 end;       { else }
         writeln;
         PrintList(output,head);
         Options(head);
         writeln;
         repeat
            write('Do you wish to enter a different list?  Y or N ---> ');
            readln(another);
            writeln;
         until another in ['Y','y','N','n'];
         if head^.ptr <> nil
            then GetRidList(head);
      until another in ['N','n'];
end;
begin
   Driver;
end.
EOF
TRSDOS Ready
```